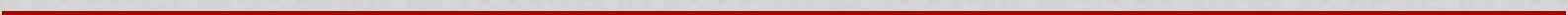


JavaScript



JavaScript

- **Linguagem de *Script* da Web**
 - Linguagem de *script* mais popular na Internet
 - Linguagem interpretada (é executada sem pré-compilação)
 - Inventada por Brendan Eich (Netscape) em 1995
 - ECMAScript é o nome oficial
 - ECMA International organization
 - ECMAScript 7.0 (última versão – Junho de 2016)



JavaScript e HTML

```
<!DOCTYPE html>
<html>
<head>
    <title>Example 2</title>
</head>

<body>
    <H1> Hello World</H1>
    <p id="demo">My First Paragraph.</p>

    <script>
        let p_elem = document.getElementById("demo");
        p_elem.innerHTML = "My First JavaScript";
    </script>

</body>
</html>
```

Elemento `<script>...</script>` é utilizado para introduzir código em JavaScript no documento HTML.

`"document.getElementById("demo")"`
- uma das formas para ir buscar ao DOM elementos HTML

Boas Práticas - o código JavaScript deve estar num ficheiro (“xpto.js”) diferente separado do HTML para permitir ler melhor o código HTML
`<script src="js/xpto.js"></script>`

Boas Práticas - evite variáveis desnecessárias de modo a aumentar o desempenho do código. Exemplo:
`document.getElementById("demo").innerHTML="My First JavaScript";`

JavaScript: Tipos Primitivos

- **Tipos de dados Primitivos**
 - number
 - string
 - boolean
 - null
 - undefined
- Todo o que não é primitivo é do tipo “object”
- Variáveis do tipo **number** são representados por **Floats de 64bits**

JavaScript: Variáveis

■ Declaração e Inicialização

```
let n;  
let carname = "Volvo";  
  
console.log(n);  
console.log(typeof(n));  
console.log(carname);  
console.log(typeof(carname));
```

```
>> undefined  
>> "undefined"  
>> Volvo  
>> "string"
```

```
let firstName = "",  
lastName = "",  
price = 0,  
discount = 0,  
fullPrice = 0,  
myArray = [],  
myObject = {};
```

Boas Práticas

- 1 - Evitar as variáveis globais (todos os tipos, objetos e funções). Variáveis globais podem criar problemas quando utilizamos outros scripts que podem ter variáveis com os mesmos nomes.
- 2 - Declarar sempre as variáveis com a keyword “let”.
- 3 - Dentro das funções criar sempre variáveis locais com a keyword “let”. Não o fazendo a variável torna-se global.
- 4 - Inicializar sempre as variáveis quando são declaradas.
- 5 - Declarar sempre as variáveis no topo das funções

JavaScript: *let*, *const* e *var*

- **const** is just like **let** but immutable
 - Declarations with **var** should be avoided or used carefully
 - **const** is very widely used with immutability being very popular
-

JavaScript: Tipos Dinâmicos

- Variável pode conter diferentes tipos de dados

```
let x;  
console.log(typeof x);  
console.log(x);  
x = 5;  
console.log(typeof x);  
console.log(x);  
x = "John";  
console.log(x);  
console.log(typeof x);
```

>> undefined
>> "undefined"

>> "number"
>> 5

>> "John"
>> "string"

Boas Práticas

- 1 - Evitar utilizar a mesma variável com tipos diferentes

JavaScript: Representação de Números

■ Numbers

```
let n1 = 1;  
let n2 = 1.5;
```

```
let n3 = 0o100;           // 0o100 – octal; 0b100 – binário  
console.log(n3);          >> 64
```

```
let n4 = 0xFF;  
console.log(n4);          >> 255
```

```
let n5 = 1e1;  
let n6 = 2e+3;  
let n7 = 2e-3;
```

```
let n8 = Infinity;  
let n10 = 6/0;  
console.log(typeof n10);  >> "number"  
console.log(n10);         >> Infinity
```

JavaScript: Exemplos de Variáveis

//Strings

```
let cname = "Volvo XC60";
let xname = `Volvo XC60`;
```

Strings são imutáveis: não podem ser alteradas.

//Booleans

```
let x = true;
let y = false;
```

//Arrays

```
let cars = ["Saab","Volvo","BMW"];
cars[3] = "Citroen";
console.log(cars);
```

Qual o tipo das variáveis “array”?

```
let cars1 = [];
cars1[0] = "Saab";
cars1[1] = "Volvo";
cars1[2] = "BMW";
cars1[3] = "Citroen";
console.log(cars1);
```



JavaScript: Operações com Variáveis

```
x = 5 + 7;          // x.valueOf() is 12, typeof x is a number  
x = 5 + "7";        // x.valueOf() is 57, typeof x is a string  
x = "5" + 7;        // x.valueOf() is 57, typeof x is a string  
x = 5 - 7;          // x.valueOf() is -2, typeof x is a number  
x = 5 - "7";        // x.valueOf() is -2, typeof x is a number  
x = "5" - 7;        // x.valueOf() is -2, typeof x is a number  
x = 5 - "x";        // x.valueOf() is NaN, typeof x is a number  
x = '5' + 5 * 5;    // '525'
```

Em algumas operações matemáticas o JavaScript pode converter números em strings

JavaScript: Template Literal

- **New syntax to create strings**

```
 ${a_variable}
```

- **Embed expressions into strings (not quote, not apostrophe but backtick)**

```
let v = 'test';
let str = `something ${v}`;
console.log(str); //something test
```

```
function foo() {
    return true;
}
```

```
const v1 = `something ${1+2+3}`; //something 6
const v2 = `something ${foo() ? "x" : "y"}`; //something x
```

JavaScript: Arrays

■ Arrays

```
let cars = new Array("Saab", "Volvo", "BMW");
let cars = ["Saab", "Volvo", "BMW"];

let cars= new Array();
cars[0] = "Saab";
cars[1] = "Volvo";
cars[2] = "BMW";

let cars = [];
cars.push("Saab");
cars.push("Volvo");
cars.push("BMW");
```

■ Métodos

- pop() - remove o último elemento
- push() - adiciona um elemento no fim
- splice() - adiciona/remove elementos numa posição
- sort() - ordena os elementos
- reverse() - inverte a ordem dos elementos
- ...

JavaScript: Objetos

```
let Person = {firstname:"John", lastname:"Doe", id:5566};  
  
let Person = {  
    firstname : "John",  
    lastname   : "Doe",  
    id         : 5566  
};  
  
let namel = Person.lastname;  
let namef = Person["firstname"];  
console.log(namel);           // 'Doe'  
console.log(namef);           // 'John'  
console.log(typeof (Person)); // object
```

■ Operador “new”

```
let carname = new String;  
let x       = new Number;  
let y       = new Boolean;  
let cars    = new Array;  
let Person  = new Object;
```

Evitar criar objetos dos **tipos primitivos** com o “new” porque tornam a execução mais lenta

JavaScript: ES6 Classes

```
class Person {  
    constructor(firstname, lastname, age, eyecolor) {  
  
        this.firstname = firstname;  
        this._lastname = lastname;  
        this.ag = age;  
        this.eyecolor = eyecolor;  
    }  
  
    walk () {  
        console.log(this.firstname + ' is walking.');  
    }  
}  
  
let bob = new Person("Bob", "Rilly", "20", "green");  
bob.walk(); // Outputs 'Bob is walking.'
```

Keyword **constructor** para definir o construtor da classe. Pode incluir métodos necessários para construir os objetos

Métodos da classe designados por métodos **prototype**

Propriedades e métodos públicos
Utilizar o operador **this** dentro do construtor para os declarar e em toda a classe para os executar

JavaScript: Regras de Sintaxe

■ Case Sensitive

```
document.getElementById("demo").innerHTML  
≠  
document.getElementByID("demo").innerHTML
```

• Quebra de linhas

```
document.write("Hello \  
World!");
```

~~```
document.write \
("Hello World!");
```~~

## JavaScript: IF ... ELSE

### ■ IF...ELSE

```
if (condition1) {
 code to be executed if condition1 is true
}
else if (condition2) {
 code to be executed if condition2 is true
}
else {
 code to be executed if condition1 and condition2 are not
true
}

let x = 5;
if (x < 5) {
 console.log ('1234');
} else if (x >= 10) {
 console.log ('101122..');
} else {
 console.log ('56789');
}
```

---

# JavaScript: Operadores de Comparação

| Operador           | Descrição                            |                                                    |
|--------------------|--------------------------------------|----------------------------------------------------|
| <code>==</code>    | <b>Igual a</b>                       | Ex: <code>x = 5; if (x == 8) =&gt; false</code>    |
| <code>====</code>  | <b>Exactamente igual a</b>           | Ex: <code>x = 5; if (x === "5") =&gt; false</code> |
| <code>!=</code>    | <b>Diferente de</b>                  | Ex: <code>x = 5; if (x != 8) =&gt; true</code>     |
| <code>!==</code>   | <b>Diferente de (incluindo tipo)</b> | Ex: <code>x = 5; if (x !== "5") =&gt; true</code>  |
| <code>&gt;</code>  | <b>Maior que</b>                     | Ex: <code>x = 5; if (x &gt; 8) =&gt; false</code>  |
| <code>&lt;</code>  | <b>Menor que</b>                     | Ex: <code>x = 5; if (x &lt; 8) =&gt; true</code>   |
| <code>&gt;=</code> | <b>Maior ou igual a</b>              | Ex: <code>x = 5; if (x &gt;= 8) =&gt; false</code> |
| <code>&lt;=</code> | <b>Menor ou igual a</b>              | Ex: <code>x = 5; if (x &lt;= 8) =&gt; true</code>  |

Comparações “strict”: operadores “`====`” e “`!==`”. Elementos têm de ser exatamente iguais, isto é, do mesmo tipo.  
Boas Práticas – devemos utilizar este operadores para evitar erros.

## JavaScript: Operadores Lógicos e Condicionais

### ■ Lógicos

| Operador | Descrição |                                         |          |
|----------|-----------|-----------------------------------------|----------|
| &&       | “And”     | Ex: x = 5; y = 3; if (x < 10 && y > 1)  | => true  |
|          | “Or”      | Ex: x = 5; y = 3; if (x == 4    y == 4) | => false |
| !        | “Not”     | Ex: x = 5; y = 3; if !(x == y)          | => true  |

### ■ Condisional

Variavelname=(*condição*)? valor1:valor2

**voteable**=(age<18)? "Too young": "Old enough";



## JavaScript: SWITCH

### ■ SWITCH

```
switch(n) {
 case 1:
 execute code block 1
 break;
 case 2:
 execute code block 2
 break;
 default:
 code to be executed if n is different from case 1 and 2
}
```

```
let x = 10;
switch(x) {
 case "10": alert("Hello");
 default : alert("Strict Comparison")
}
```

**NÃO ESQUECER:** “switch” utiliza comparações “strict”.

## JavaScript: Repetições – FOR

### ■ FOR

```
for (let v = Startvalue; v <= Endvalue; v = v+Increment)
{
 code to be executed
}
```

### ■ FOR IN

```
let txt = [];
let Person = {fname:"John",lname:"Doe",age:25};

for (let x in Person) {
 txt = txt + Person[x];
}

txt => JohnDoe25
```

## JavaScript: “while” e “do... while”

### ■ **while**

```
while (condition) {

 code block to be executed

}
```

### ■ **do while**

```
do {

 code block to be executed

} while (condition);
```

---

## JavaScript: Functions

### Function

```
function functionname(var1,var2,...,varX)
{
 some code
}
```

The diagram illustrates the execution flow of a JavaScript function. It shows two parts of an HTML file: the `<head>` section and the `<body>` section. In the `<head>` section, there is a `<script>` block containing the function definition:

```
<head>
 <script>
 function product(a,b) {
 return a * b;
 }
 </script>
</head>
```

An arrow points from this code to a yellow box labeled "Definição da função". In the `<body>` section, there is also a `<script>` block containing the function call:

```
<body>
 <script>
 document.write("<h1> Function Example </h1>");
 document.write(product(4,3));
 </script>
</body>
```

An arrow points from this code to another yellow box labeled "Execução da função para os valores passados como parâmetros".

## JavaScript: Ponto e vírgula

```
if (x === 19);
{
 // code block
}
```

Código executado  
independentemente do valor de 'x'

```
function myFunction(a) {
 let power = 10
 return a * power
}
```

```
function myFunction(a) {
 let power = 10;
 return a * power;
}
```

Mesmo código. JavaScript  
fecha a instrução no fim da  
linha automaticamente.

```
function myFunction(a) {
 let
 power = 10;
 return
 a * power;
}
```

**ATENÇÃO:** nesta situação é colocado  
automaticamente um ponto e vírgula no fim do  
return. A última instrução não é executada

## JavaScript: Eventos

### Sintaxe

```
<element eventname="SomeJavaScriptCode">

<head>
 <script>
 function myFunction() {
 document.getElementById("demo").innerHTML="Hello World";
 }
 </script>
</head>

<body>
 <p>Click the button to trigger a function.</p>
 <button onclick="myFunction()">Click me</button>
 <p id="demo"></p>
</body>
```

Evento “onclick” associado  
a um “button” definido por  
elemento HTML

## JavaScript: Eventos – Mouse

### ■ Mouse

Evento	Descrição
onclick	Ocorre quando o utilizador clica num elemento
ondblclick	Ocorre quando o utilizador faz <i>double-click</i> num elemento
onmousedown	Ocorre quando o utilizador pressiona um botão sobre um elemento
onmousemove	Ocorre quando o <i>mouse</i> está em movimento enquanto estiver sobre um elemento
onmouseover	Ocorre quando o <i>mouse</i> se move sobre um elemento
onmouseout	Ocorre quando o utilizador move o <i>mouse</i> para fora de um elemento
onmouseup	Ocorre quando o utilizador libertar um botão sobre um elemento

## JavaScript: Eventos – Keyboard e Window

### ■ Keyboard

Evento	Descrição
onkeydown	Ocorre quando o utilizador esta a pressionar uma tecla
onkeypress	Ocorre quando o utilizador pressiona uma tecla
onkeyup	Ocorre quando o utilizador liberta (deixa de pressionar) uma tecla

### ■ Window/Object

Evento	Descrição
onload	Ocorre quando um documento ou um objecto foi carregado (loaded)
onerror	Ocorre quando uma imagem não foi carregada de forma adequada
onscroll	Ocorre quando é feito scroll num documento

## JavaScript: Gestão de Erros

```
<head>
 <script>
 let txt = "";
 function message() {

 try {
 adddalert("Welcome guest!");
 }

 catch(err) {
 txt = "There was an error on this page.\n\n";
 txt += "Error description: " + err.message + "\n\n";
 txt += "Click OK to continue.\n\n";
 alert(txt);
 }
 }

 </script>
</head>
<body>
 <input type="button" value="View message" onclick="message()">
</body>
```

Gestão de erros idêntica a outras linguagens de programação